V&V and Software Engineering: What Have We Learned From DARPA's HPCS Program

Lawrence Votta

(larry@brincos.com)

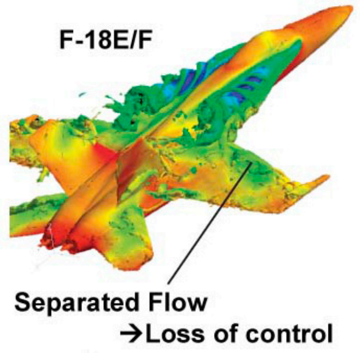
Brincos Inc., Seattle

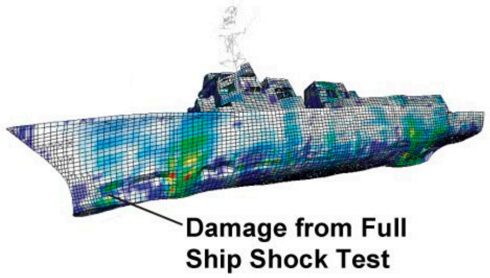
August 2011

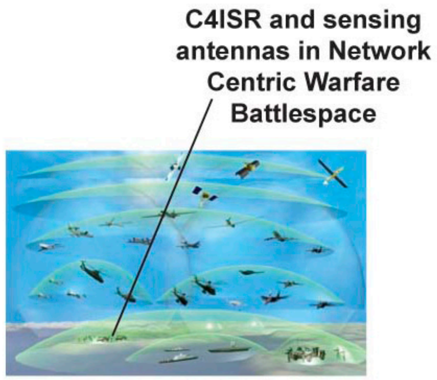
Computational Research and Engineering Acquisition Tools and Environments

CREATE
Computational Research & Engineering for Acquisition Tools & Environments

- $360M 12-year program to develop & deploy 3 computational engineering tool sets for acquisition engineers
- **Air Vehicle** design tools: Aerodynamics, **air frame, propulsion, control,** early rapid design
- **Ship** design tools: Early-stage design, shock damage and hydrodynamics performance
- **RF Antenna** design tools: RF Antenna performance and integration with platforms
- **MG** Computational Infrastructure






C4ISR and sensing antennas in Network Centric Warfare Battlespace

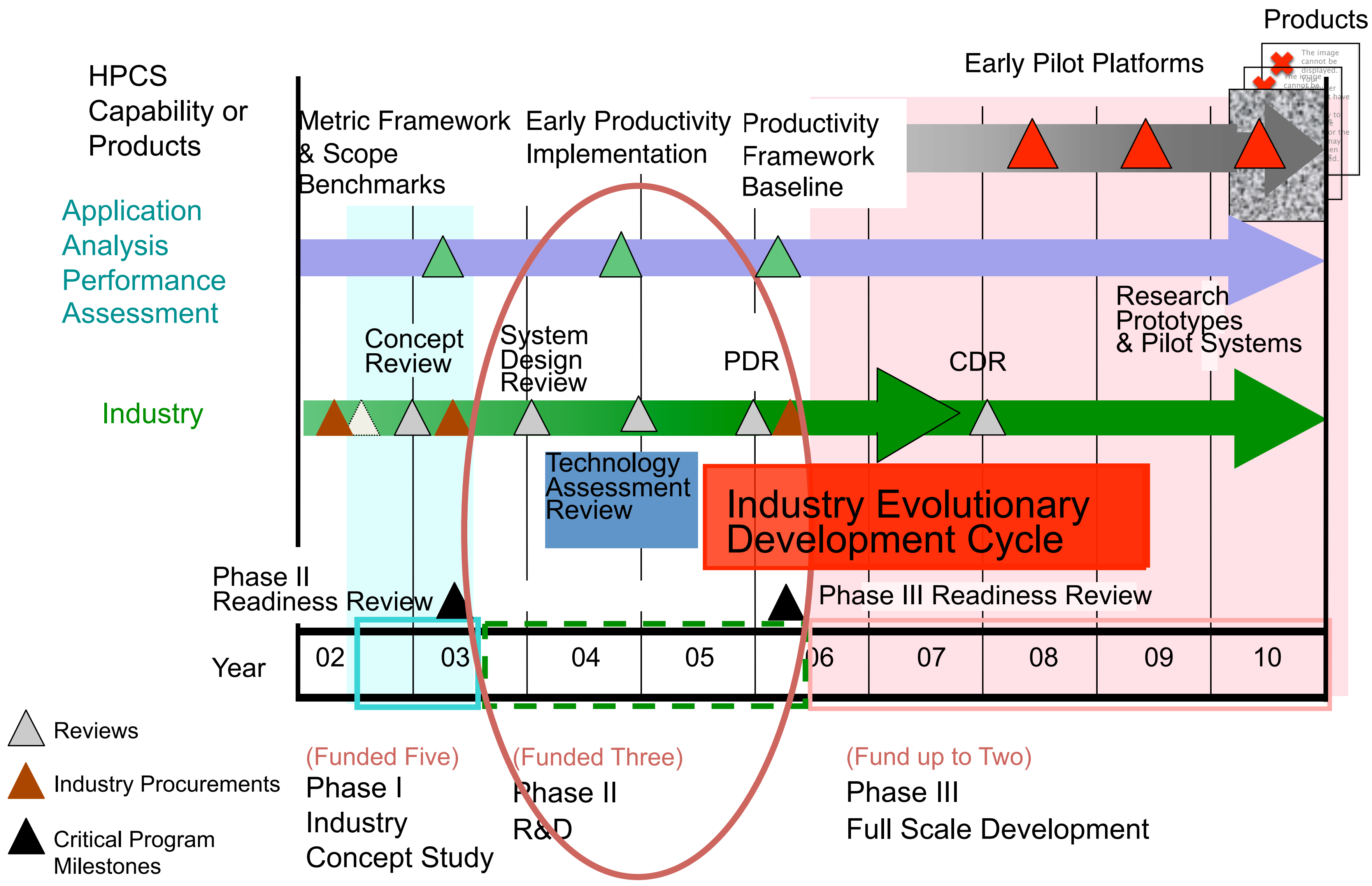
Sun’s HPCS Productivity Team

- Russ Brown – Computational Scientist
- Stuart Faulk – Software Engineering, Computer Science
- Eugene Loh – Computational Scientist
- Declan Murphy – Maintenance, Administration Arch
- Susan Squires – Cultural Anthropologist
- Walter Tichy – Computer Science, Software Engineer
- Michael L. Van De Vanter – Software Tools
- Christopher Vick – Software System Architect
- Lawrence G. Votta – Computer Arch, Software Engineer
- Alan Wood – Fault Tolerance Architect

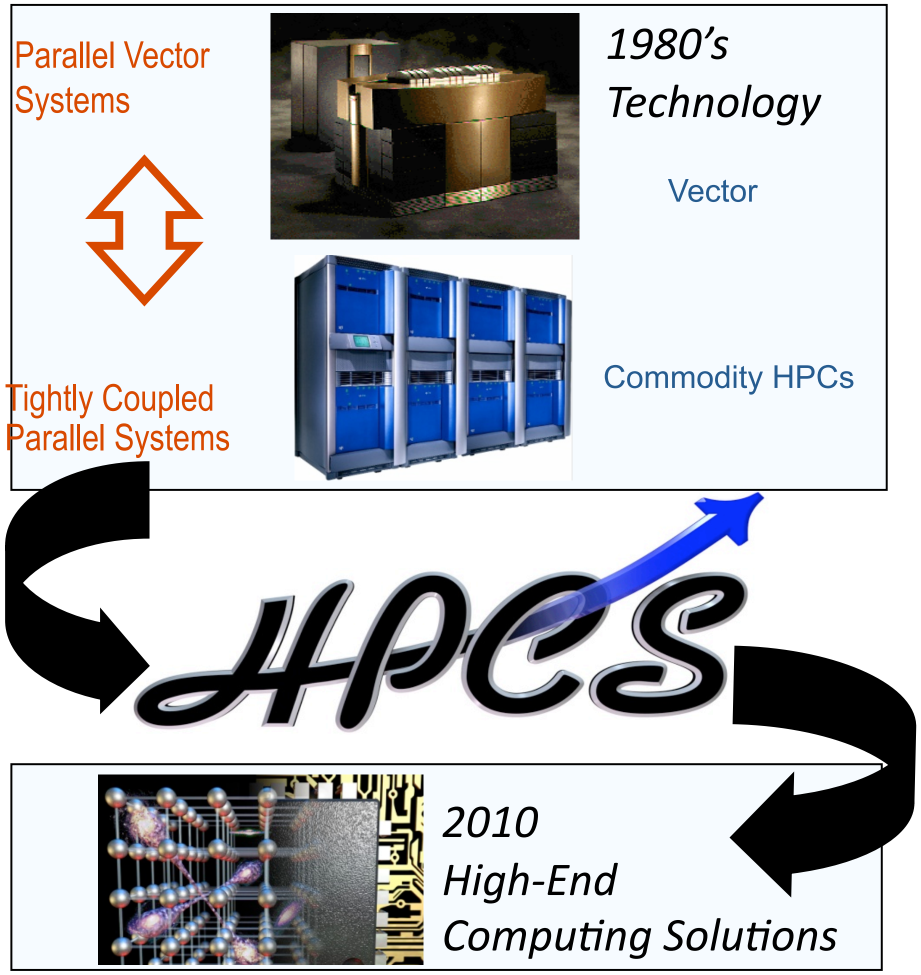
Robust Productivity Results from Software Engineering

- Abstraction (Dijkstra & Parnas)
  - Separation of Concerns
  - Information Hiding, Encapsulation
- Automation
  - Programming Languages
  - Operating Systems
  - Tools
- Size Matters
  - Maintenance and Evolution Effort
  - Boehm's Software Engineering Economics
  - User community

HPCS Program Phases I - III


Products
Early Pilot Platforms
HPCS Capability or Products
Application Analysis Performance Assessment
Industry
Metric Framework & Scope Benchmarks
Early Productivity Implementation
Productivity Framework Baseline
Concept Review
System Design Review
PDR
CDR
Research Prototypes & Pilot Systems
Technology Assessment Review
Industry Evolutionary Development Cycle
Phase II Readiness Review
Phase III Readiness Review
Year
02
03
04
05
06
07
08
09
10
Reviews
Industry Procurements
Critical Program Milestones
(Funded Five)
Phase I Industry Concept Study
(Funded Three)
Phase II R&D
(Fund up to Two)
Phase III Full Scale Development

Vision: Focus on the Lost Dimension of HPC
“User & System Efficiency and Productivity”


Parallel Vector Systems
1980’s Technology
Vector
Commodity HPCs
Tightly Coupled Parallel Systems
HPCS
2010 High-End Computing Solutions

Moore’s Law Double Raw Performance every 18 Months

New Goal: Double Value Every 18 Months

Industry Accepted Metrics Drive End Products

HPCS Phase II Teams

Industry:

IBM
PI: Elnozahy

Sun microsystems
PI: Votta

CRAY
PI: Smith

Mission Partners:

Office of Science
U.S. Department of Energy

NASA

National Nuclear Security Administration

DoD HPC Modernization Program

Productivity Team (Lincoln Lead)

MIT Lincoln Laboratory
PI: Kepner

ISI
PI: Lucas

University of Maryland
PI: Basili

UCSD
PI: Benson & Snavely

ICL UT
PI: Dongarra

MITRE
PI: Koester

Oak Ridge, Los Alamos National Laboratory, Argonne, Berkeley Lab
PIs: Vetter, Lusk, Post, Bailey

UCSB, LCS, Ohio State, CodeSourcery
PIs: Gilbert, Edelman, Ahalt, Mitchell

Motivation

- Auto Crash Models
- Proctor Gamble Packaging
- Weather Modeling and Prediction
- Cell Phone Network Layout
- Computational Marketing (Costco, Walmart, …)
- Entertainment – Motion Pictures
- Swim Suit Design
- Tire Design and Optimization

Japan Held Nuclear Data, Leaving Evacuees in Peril

Ko Sasaki for The New York Times

Contaminated soil at a school in Koriyama, Japan.

By NORIMITSU ONISHI and MARTIN FACKLER

Published: August 8, 2011

FUKUSHIMA, Japan — The day after a giant tsunami set off the continuing disaster at the Fukushima Daiichi nuclear plant, thousands of residents at the nearby town of Namie gathered to evacuate.

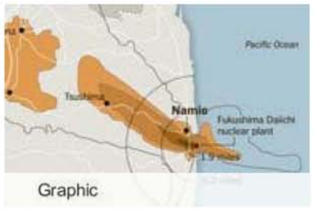
Given no guidance from Tokyo, town officials led the residents north, believing that winter winds would be blowing south and carrying away any radioactive emissions. For three nights, while hydrogen explosions at four of the reactors spewed radiation into the air, they stayed in a district called Tsushima where the children played outside and some parents used water from a mountain stream to prepare rice.

The winds, in fact, had been blowing directly toward Tsushima — and town officials would learn two months later that a government computer system designed to predict the spread of radioactive releases had been showing just that.

But the forecasts were left unpublicized by bureaucrats in Tokyo, operating in a culture that sought to avoid responsibility and, above all, criticism. Japan's political leaders at first did not know about the system and later played down the data, apparently fearful of having to significantly enlarge the evacuation zone — and acknowledge the accident's severity.

"From the 12th to the 15th we were in a location with one of the highest levels of radiation," said Tamotsu Baba, the

Multimedia



An Early Forecast of Radiation

Related

Fatal Radiation Level Found at Japanese Plant (August 2, 2011)

Japanese Find Radioactivity on Their Own (August 1, 2011)

Times Topic: Japan — Earthquake, Tsunami and Nuclear Crisis (2011)

Enlarge This Image

Ko Sasaki for The New York Times

PREVENTIVE MEASURES Officials in

RECOMMEND
TWITTER
COMMENTS (23)
E-MAIL
PRINT
SINGLE PAGE
REPRINTS
SHARE

ANOTHER EARTH
WATCH TRAILER

Find the Best Fuel Efficient Cars. Check out Reviews & Listings-Today!
worldclassdriving.com

We Buy Salvaged Cars
Sell Your Car Today. Get Cash Fast! Request Free Online Offer Now.
Copart.com

Advertise on NYTimes.com

TimesLimited E-Mail

Sign up to receive exclusive products and experiences featuring NYTimes.com's premier advertisers.
votta@alum.mit.edu Sign Up
Change E-mail Address | Privacy Policy

MOST E-MAILED MOST VIEWED

1. OP-ED COLUMNIST
Credibility, Chutzpah and Debt

2. OPINION
What Happened to Obama?

3. Virginia Heffernan: Education Needs a Digital-Age Upgrade

4. By Helping a Girl Testify at a Rape Trial, a Dog Ignites a Legal Debate

5. On Idyllic Cape Cod, Growing Drug Problem Fuels a Rise in Property Crimes

6. The Phantom Menace of Sleep Deprived Doctors

7. Overriding a Key Education Law

LIVES RESTORED

Koriyama, Japan, removed surface soil from its schools for fear of radiation contamination and imposed tougher inspection standards than those set by the country's education officials.

mayor of Namie, which is about five miles from the nuclear plant. He and thousands from Namie now live in temporary housing in another town, Nihonmatsu. "We are extremely worried about internal exposure to radiation."

The withholding of information, he said, was akin to "murder."

In interviews and public statements, some current and former government officials have admitted that Japanese authorities engaged in a pattern of withholding damaging information and denying facts of the nuclear disaster — in order, some of them said, to limit the size of costly and disruptive evacuations in land-scarce Japan and to avoid public questioning of the politically powerful nuclear industry. As the nuclear plant continues to release radiation, some of which has slipped into the nation's food supply, public anger is growing at what many here see as an official campaign to play down the scope of the accident and the potential health risks.

Seiki Soramoto, a lawmaker and former nuclear engineer to whom Prime Minister Naoto Kan turned for advice during the crisis, blamed the government for withholding forecasts from the computer system, known as the System for Prediction of Environmental Emergency Dose Information, or Speedi.

"In the end, it was the prime minister's office that hid the Speedi data," he said. "Because they didn't have the knowledge to know what the data meant, and thus they did not know what to say to the public, they thought only of their own safety, and decided it was easier just not to announce it."

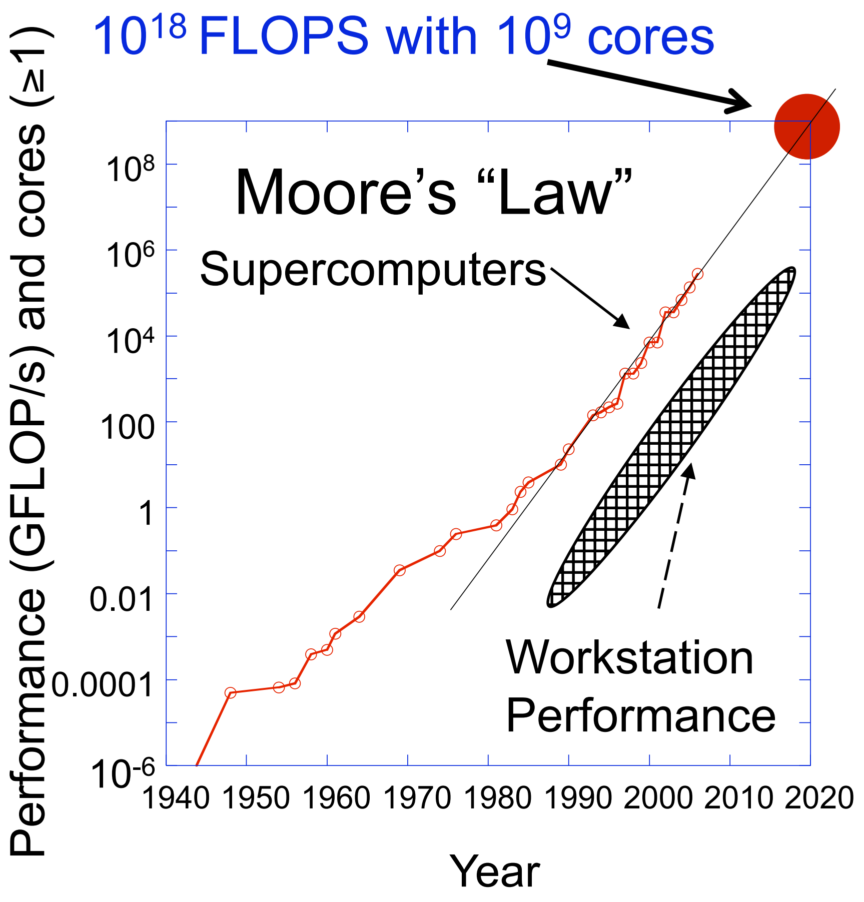
In an interview, Goshi Hosono, the minister in charge of the nuclear crisis, dismissed accusations that political considerations had delayed the release of the early Speedi data. He said that they were not disclosed because they were incomplete and inaccurate, and that he was presented with the data for the first time only on March 23.

"And on that day, we made them public," said Mr. Hosono, who was one of the prime minister's closest advisers in the early days of the crisis before being named nuclear

Next Generation Computers Offer Unprecedented Power to Address Important Problems

- Next generation computers (2020) will provide exciting opportunities to develop and deploy very powerful application codes:
  - Utilize accurate solution methods
  - Include all the effects we know to be important
  - Model a complete system
  - Complete parameter surveys in hours rather than days to weeks to months
- In ~ 10 years, workstations will be as powerful as today's supercomputers
- Need to develop codes that exploit this capability
  - Multi-Physics codes that can scale from the present (~ 10^2 cores) to the future (10^6 to 10^9 cores)

Computing Power For The World's Fastest Computers



Productivity Problem

Today
100 – 1000
Processors

Future
10K – 100K
Processors

HPCS Goal:
10X Productivity
Improvement

Scientific and Engineering Programming

- *It's about the science and engineering*
- *Scientific and engineering codes are expensive*
- *Codes live a long time*
- *Performance really matters*
- *Hardware platforms change often*
- *Its all Fortran 77 and C++*
- *Hardware cost dominate*
- *Ports are frequent*
- *V&V expensive*

Productivity Problem = Computational Science Crises?

- Performance Challenge—Designing and building high performance Computers
- Programming Challenge—Programming for Complex Computers
  - Rapid code development
  - Optimize codes for good performance
- Prediction Challenge—Developing predictive codes with complex scientific models
  - Develop codes that have reliable predictive capability
    - Verification
    - Validation
    - Code Project Management and Quality

Productivity Crises = Gridlock

- Symptoms
  - Long and troubled software developments
  - Dysfunctional market to support tools
  - Scientific results based on software
- Scientific programmers
  - *"… computer scientist don't address our needs."*
  - *"… there isn't enough money."*
- Communication Gap

How to Study Productivity

- Embrace the broadest possible view of productivity including human tasks, skills, motivations, organizations and culture.
- Put the investigation on the soundest possible scientific basis, drawing on both physical and social science methods.

➢ Three Stage Research Framework

How to Study Productivity: Three Stage Research Framework

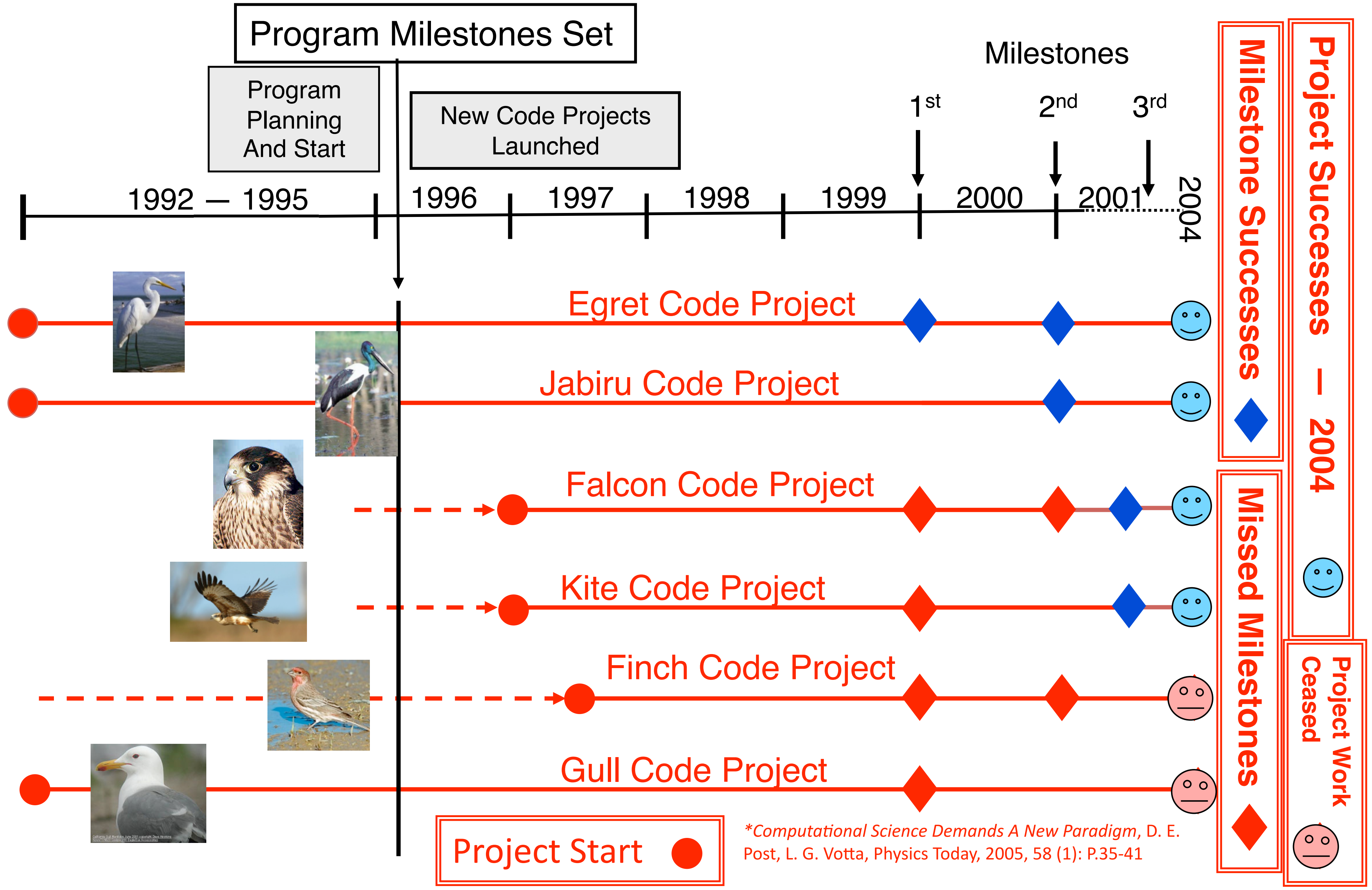
STAGE	**Explore and Discover** →	**Test and Define** →	**Evaluate and Validate**
GOALS	**Develop Hypotheses**	**Test and Refine Models**	**Replicate and Validate Findings**
METHODS	**Qualitative**	**Qualitative and Quantitative**	**Quantitative**

How to Study Productivity: Three Stage Research Framework

- Stage 1: Explore and Discover
  - Observe phenomena under study and acquire insights necessary for hypothesis generation.
- Stage 2: Test and Define
  - Rigor is added that produce insights, supply concrete data for model refinement and lead to deeper understanding.
- Stage 3: Evaluate and Validate
  - Quantitative techniques refine the resuls and validate the outcomes.

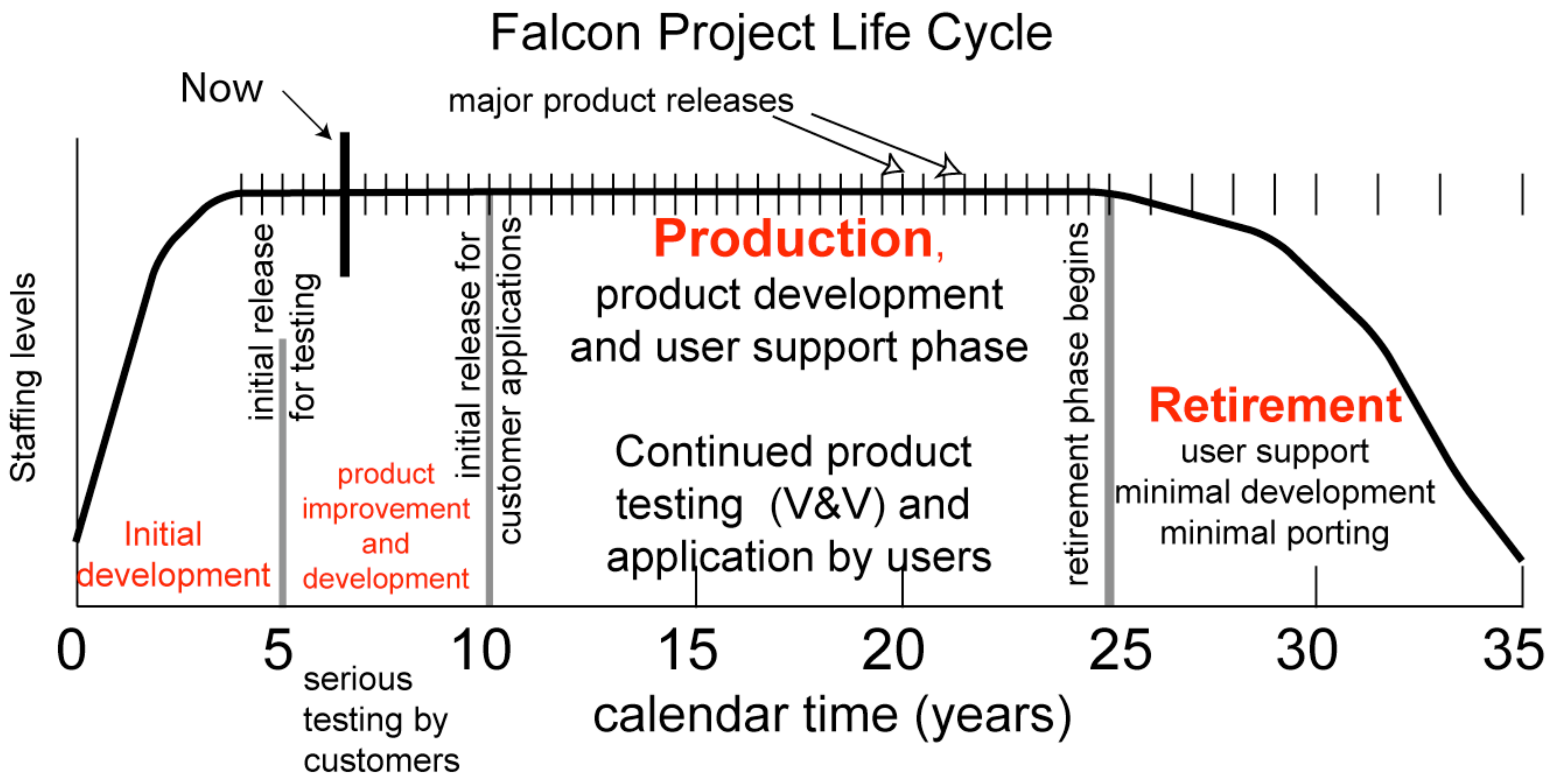
Risk Mitigation Often Requires Redundant Projects*.

Code Project Schedule For Six Large-scale Physics Codes



**Computational Science Demands A New Paradigm*, D. E. Post, L. G. Votta, Physics Today, 2005, 58 (1): P.35-41

Typical Application Lifecycle



Productivity Gridlock = Bottlenecks

Programming Workflow

- Developing correct scientific programs
- Serial optimization and tuning
- Code parallelization and optimization
- Porting

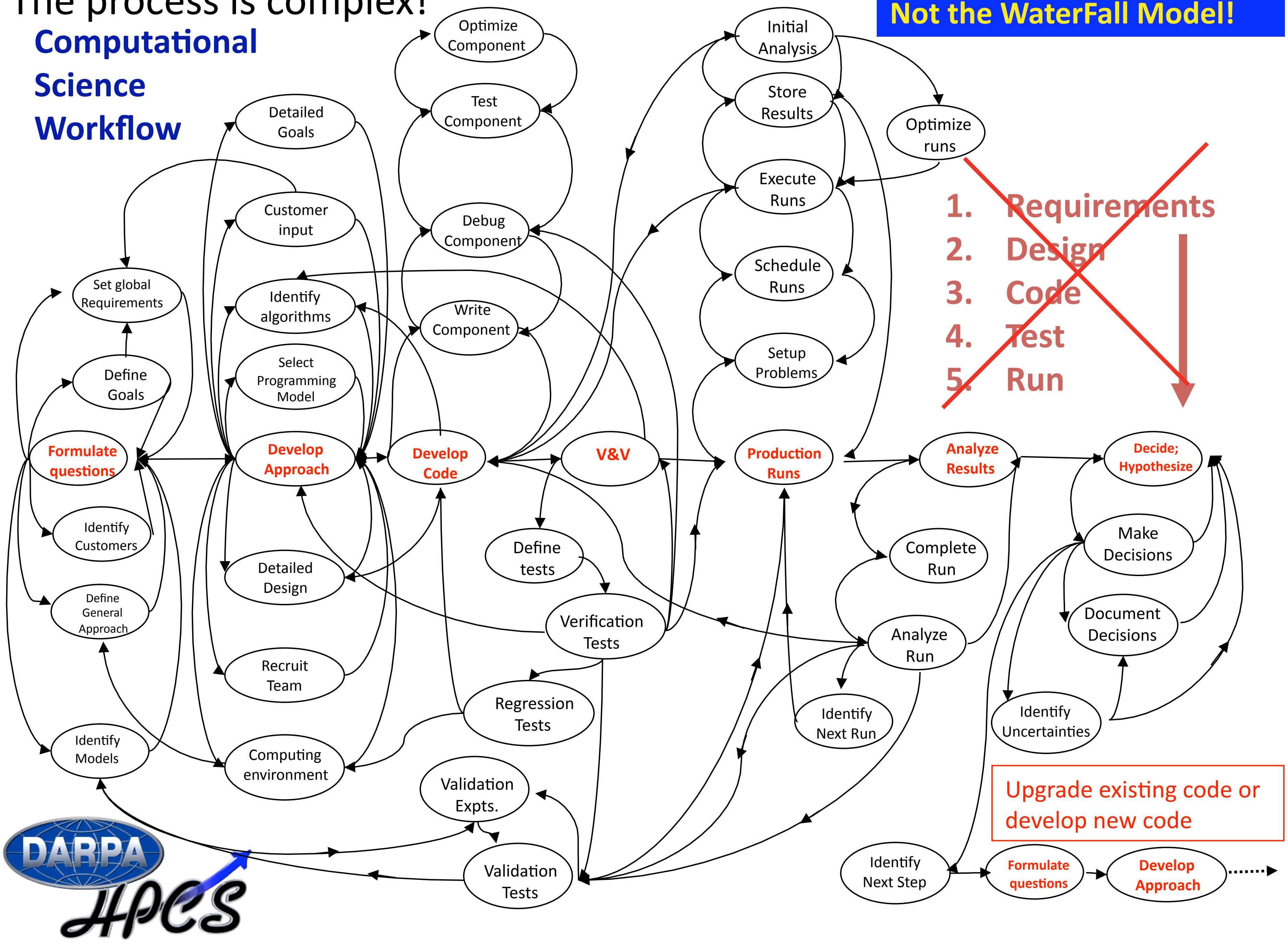
⇨ Manual Programming Effort

⇨ Expertise

The process is complex!

Computational Science Workflow

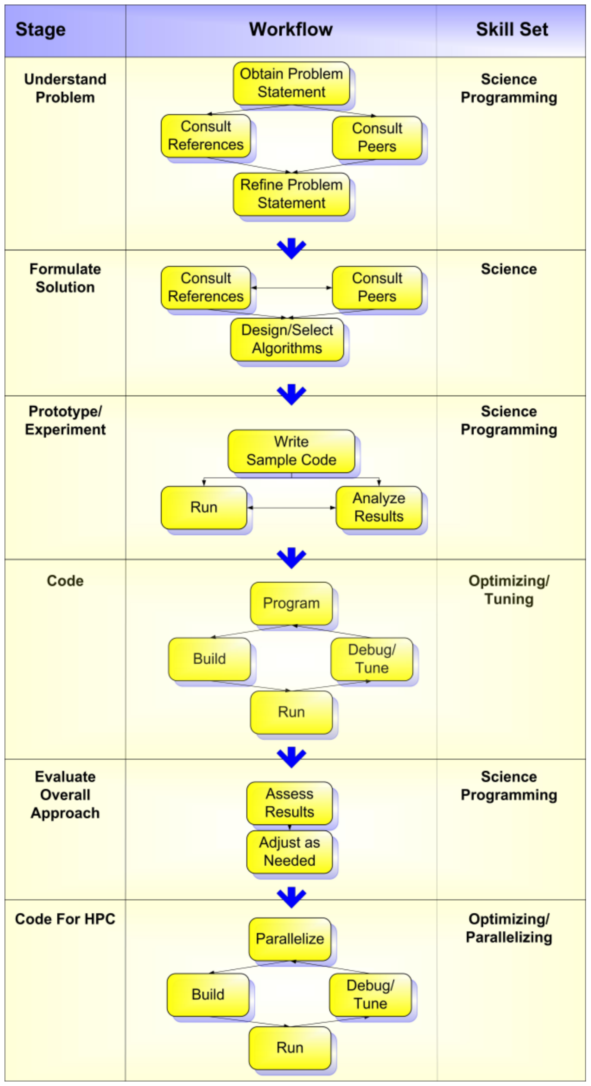
Not the WaterFall Model!



1. ~~Requirements~~
2. ~~Design~~
3. ~~Code~~
4. ~~Test~~
5. ~~Run~~

Upgrade existing code or develop new code

Programming Workflow



Productivity Bottleneck
Expertise Gap

- Four Distinct Skill Sets
  - Domain Science
  - Scientific Computing
  - Scaling
  - Management
- The skills are only useful when they are synchronized through communication and collaboration or exist in one person.
  - 2 skill sets rare; 3 skill very rare; 4 …

Where Are the Tools?
Abstraction & Automation - Gaps

Scientist Tool Complaints

- “hard to learn”
- “don’t scale”
- “differ across platforms”
- “poorly supported”
- “too expensive”

Where Are the Tools? Abstraction & Automation - Gaps

Contributing Factors

- General computing Integration Developing Environments' have different assumptions.
- Lifetime of codes slows evolution of tools, necessitating long life cycles.
- Field is small and specialized.
- Investment in tools is insufficient.
- Business demands may remove important tools from the market.

Is It Nature or Is It Software?

- Trust in the validity of computational outcomes a key productivity issue.
- How do scientist build confidence in their codes?
  - "... looked right"
  - "... 4 or 5 years to get some confidence in code."
- Scientist manage threats to validity in experimental designs but not in codes.
  - Worked needed here.

Breaking the Gridlock – Can Software Engineering Help?

- Software Engineering
  - Automation
  - Abstraction
  - Measurement
- Scientific Programming
  - Investment
  - Modernization

Promising Experiments 1 – Breaking the Myths

- Computational Science Myth
  - Can not achieve performance using high level language.
  - Our experiments with NAS benchmarks, GTC (computational CFD of heat loss in tomacks) and Amber (electrostatic potential of complex proteins) show that 10X (less code) productivity at same performance.
- Computer Science Myth
  - Implement serial version then parallelize for multicore and clusters.
  - Experiments with programming teams indicate this is the wrong strategy – suboptimal solutions achieved.

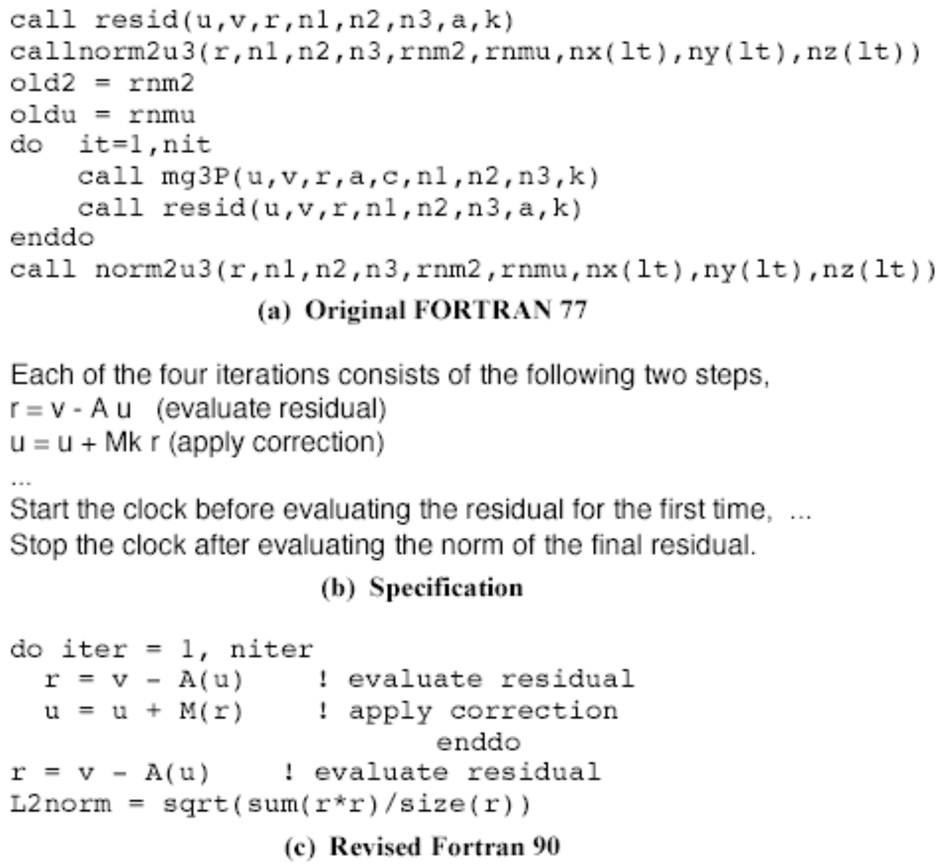
Promising Experiments 2 – Breaking the Computational Myths

- NAS Parallel Benchmarks – port F77 versions to Fortran 90
  - Removed specialized code for distributed memory
  - Removed source level optimizations
  - Exploited abstractions provided by Fortran 90 a superset of F77.
  - Remove code not portable.
- Result – reduced source code by ~10x
  - Maintenance – lifetime cost scale with size
  - Portability – removed code not portable
  - Correctness – see next slide.

Perfective Code Maintenance

- Remove specialized code for distributed memory
- Remove source level optimizations
  - e.g., Loop unrolling, intermediate values
- Exploit abstractions supported by Fortran 90
- Seek specification-code alignment
- Remove code known not to be portable

Promising Experiments – Breaking the Computational Myths



```
call resid(u,v,r,n1,n2,n3,a,k)
callnorm2u3(r,n1,n2,n3,rnm2,rnmu,nx(lt),ny(lt),nz(lt))
old2 = rnm2
oldu = rnmu
do  it=1,nit
    call mg3P(u,v,r,a,c,n1,n2,n3,k)
    call resid(u,v,r,n1,n2,n3,a,k)
enddo
call norm2u3(r,n1,n2,n3,rnm2,rnmu,nx(lt),ny(lt),nz(lt))
```

(a) Original FORTRAN 77

Each of the four iterations consists of the following two steps,
r = v - A u   (evaluate residual)
u = u + Mk r (apply correction)
...
Start the clock before evaluating the residual for the first time, ...
Stop the clock after evaluating the norm of the final residual.

(b) Specification

```
do iter = 1, niter
  r = v - A(u)    ! evaluate residual
  u = u + M(r)    ! apply correction
                       enddo
r = v - A(u)    ! evaluate residual
L2norm = sqrt(sum(r*r)/size(r))
```

(c) Revised Fortran 90

Figure 2. "High Productivity" code excerpt from NAS MG (timed portion)

Code Experiment Results: Code Comparisons

Code Name		Lines of Code (SLOC)		Performance Slow Down
	Before	After	Reduction	
NPB CG	839	81	10x	1x
NPB MG	1,701	150	11x	2x-6x
NPB BT	4,234	594	7x	2.7x
GTC	6,736	1,889	3.6x	2.7x
sPPM	13,606	1,358	10x	2x

A More Realistic Experiment: GTC Plasma Physics Code

- *"At first glance, I was impressed by how small and compact the code had become. I always thought that GTC was as small as it could get, but I was obviously wrong. I was also pleasantly surprised to discover that the programming language was still standard Fortran 90/95, and not a totally new language."*
- *"The new code is clear, concise, and easy to read."*
- *"The fact that all the MPI calls and OpenMP directives have been removed makes the physics represented in the code easier to follow."*
- *"[Expect a] performance hit unless the compiler can perform very good interprocedural optimization and/or automatic inlining."*

Computational Science And Engineering Has At Least Four Major Elements.

Computers	Codes	V&V	Users
Making enormous progress but at cost of complexity, particularly memory hierarchy	More complicated models + larger programming challenges	Harder due to inclusion of more effects and more complicated models	Use tools to solve problems, do designs, make discoveries
Need to reduce programming challenge	**Greatest bottleneck**	Inadequate methods, need paradigm shift	Users make connections to customers

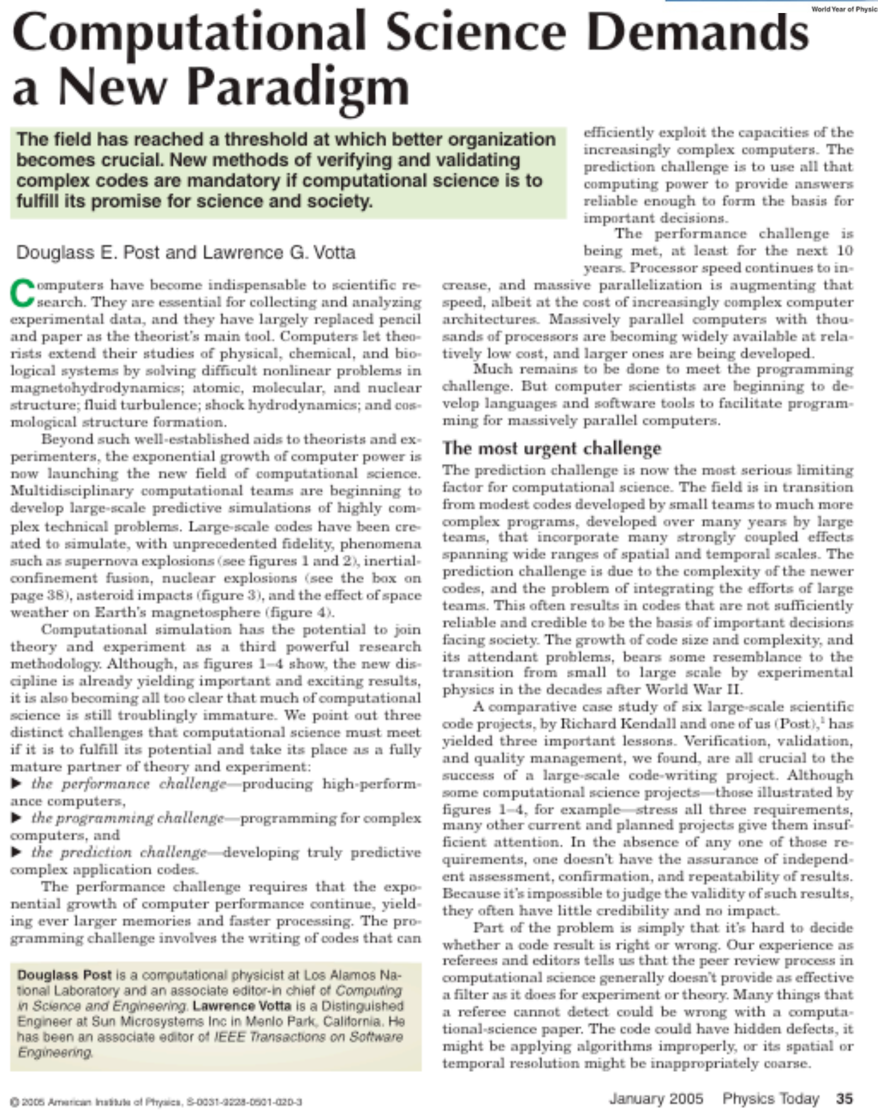
→ sponsors

- **We need to develop a total capability to solve problems, not just build codes or computers.**

Issues Summarized In January 2005 Physics Today Article*.

- Three Challenges
  - Performance Challenge
  - Programming Challenge
  - Prediction Challenge
    - Where case studies are important
- Case Studies are needed for success
  - The Scientific Method
- Paradigm shift needed
  - Computational Science moving from few effect codes developed by small teams to many effect codes developed by large teams
  - Similar to transition made by experimental science in 1930—1960
  - Software Project Management and V&V need more emphasis

*_Computational Science Demands a New Paradigm_, D.E. Post and L.G. Votta, Physics Today,**58**(1), 2005, p.35-41.
Email **post@ieee.org to get a copy.**



Computational Science Demands a New Paradigm

The field has reached a threshold at which better organization becomes crucial. New methods of verifying and validating complex codes are mandatory if computational science is to fulfill its promise for science and society.

Douglass E. Post and Lawrence G. Votta

Computers have become indispensable to scientific research. They are essential for collecting and analyzing experimental data, and they have largely replaced pencil and paper as the theorist's main tool. Computers let theorists extend their studies of physical, chemical, and biological systems by solving difficult nonlinear problems in magnetohydrodynamics; atomic, molecular, and nuclear structure; fluid turbulence; shock hydrodynamics; and cosmological structure formation.

Beyond such well-established aids to theorists and experimenters, the exponential growth of computer power is now launching the new field of computational science. Multidisciplinary computational teams are beginning to develop large-scale predictive simulations of highly complex technical problems. Large-scale codes have been created to simulate, with unprecedented fidelity, phenomena such as supernova explosions (see figures 1 and 2), inertial-confinement fusion, nuclear explosions (see the box on page 38), asteroid impacts (figure 3), and the effect of space weather on Earth's magnetosphere (figure 4).

Computational simulation has the potential to join theory and experiment as a third powerful research methodology. Although, as figures 1–4 show, the new discipline is already yielding important and exciting results, it is also becoming all too clear that much of computational science is still troublingly immature. We point out three distinct challenges that computational science must meet if it is to fulfill its potential and take its place as a fully mature partner of theory and experiment:

▶ _the performance challenge_—producing high-performance computers,

▶ _the programming challenge_—programming for complex computers, and

▶ _the prediction challenge_—developing truly predictive complex application codes.

The performance challenge requires that the exponential growth of computer performance continue, yielding ever larger memories and faster processing. The programming challenge involves the writing of codes that can efficiently exploit the capacities of the increasingly complex computers. The prediction challenge is to use all that computing power to provide answers reliable enough to form the basis for important decisions.

The performance challenge is being met, at least for the next 10 years. Processor speed continues to increase, and massive parallelization is augmenting that speed, albeit at the cost of increasingly complex computer architectures. Massively parallel computers with thousands of processors are becoming widely available at relatively low cost, and larger ones are being developed.

Much remains to be done to meet the programming challenge. But computer scientists are beginning to develop languages and software tools to facilitate programming for massively parallel computers.

The most urgent challenge

The prediction challenge is now the most serious limiting factor for computational science. The field is in transition from modest codes developed by small teams to much more complex programs, developed over many years by large teams, that incorporate many strongly coupled effects spanning wide ranges of spatial and temporal scales. The prediction challenge is due to the complexity of the newer codes, and the problem of integrating the efforts of large teams. This often results in codes that are not sufficiently reliable and credible to be the basis of important decisions facing society. The growth of code size and complexity, and its attendant problems, bears some resemblance to the transition from small to large scale by experimental physics in the decades after World War II.

A comparative case study of six large-scale scientific code projects, by Richard Kendall and one of us (Post),[1] has yielded three important lessons. Verification, validation, and quality management, we found, are all crucial to the success of a large-scale code-writing project. Although some computational science projects—those illustrated by figures 1–4, for example—stress all three requirements, many other current and planned projects give them insufficient attention. In the absence of any one of those requirements, one doesn't have the assurance of independent assessment, confirmation, and repeatability of results. Because it's impossible to judge the validity of such results, they often have little credibility and no impact.

Part of the problem is simply that it's hard to decide whether a code result is right or wrong. Our experience as referees and editors tells us that the peer review process in computational science generally doesn't provide as effective a filter as it does for experiment or theory. Many things that a referee cannot detect could be wrong with a computational-science paper. The code could have hidden defects, it might be applying algorithms improperly, or its spatial or temporal resolution might be inappropriately coarse.

Douglass Post is a computational physicist at Los Alamos National Laboratory and an associate editor-in chief of _Computing in Science and Engineering_. **Lawrence Votta** is a Distinguished Engineer at Sun Microsystems Inc in Menlo Park, California. He has been an associate editor of _IEEE Transactions on Software Engineering_.

© 2005 American Institute of Physics, S-0031-9228-0501-020-3

Concluding Remarks

- Computational Science and Engineering are here to stay.
- Do not succumb to myths!
  - Computer Science and Software Engineering is improving
  - Serial to parallel model
- Productivity crises is an expertise crises.
- V&V&UQ (and Certification) credibility is important.

BACKUPS

Bibliography - 1

1. D. F. Kelly, "A Software Chasm: Software Engineering and Scientific Computing." *IEEE Software. 24, 6 (Nov. 2007), 120-119.*
2. J. Carver, R. Kendall, S. Squires, and D. Post, "Software Development Environments for Scientific and Engineering Software: A Series of Case Studies." In *Proceedings of the 29th international Conference on Software Engineering (May 20 - 26, 2007). International Conference on Software* Engineering. IEEE Computer Society, Washington, DC, 550-559
3. S. Squires, M. Van De Vanter, and L. Votta, "Yes, There Is an 'Expertise Gap' In HPC Applications Development", *Proceedings of the Third Workshop on Productivity and Performance in High-End Computing* (PPHEC'06), Austin, TX, Feb. 12, 2006.
4. M. Van De Vanter, D. Post and M. Zosel, "HPC Needs a Tool Strategy"*, Second International Workshop on SE HPC Systems Applications, May* 2005.
5. D. Post and L. Votta, "Computational Science Demands a New Paradigm", *Physics Today, 58(1), pp. 35-41, 2005).*

Bibliography – 2

6. E. Loh, M. Van De Vanter and L. Votta. Can Software Engineering Solve the HPCS Problem? *Second International Workshop on SE HPC Sys Applications, May 2005.*
7. S. Squires, M. Van De Vanter, and L. Votta, "Software Productivity Research In High Performance Computing," Cyberinfrastructure Technology Watch (CTWatch) Quarterly: High Productivity Computing Systems and the Path Towards Usable Petascale Computing, 2(4a), November 2006, pp. 52-61. http://www.ctwatch.org/quarterly/archives/november-2006-a/
8. C. Holland, DoD Research and Development Agenda for High Productivity Computing Systems (White Paper), Pentagon, Washington, DC, June 2001.
9. Goldberg, D. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys 23, 1 (Mar. 1991), 5-48.*
10. *See Tichy, et al at <http://www.ipd.uni-karlsruhe.de/ multicore/research/download/Pankratius-bzip-multicore.pdf>*

Bibliography - 3

11. S. Faulk, S. Squires, M. Van De Vanter, C. Vick, L. Votta and A. Wood, “Productive Petascale Computing: Requirements, Hardware, and Software”. Sun Microsystems Technical Report-2009-183.
12. See E. Loh, “The Ideal Programming Language” at http://queue.acm.org/detail.cfm?id=1820518, 2010.

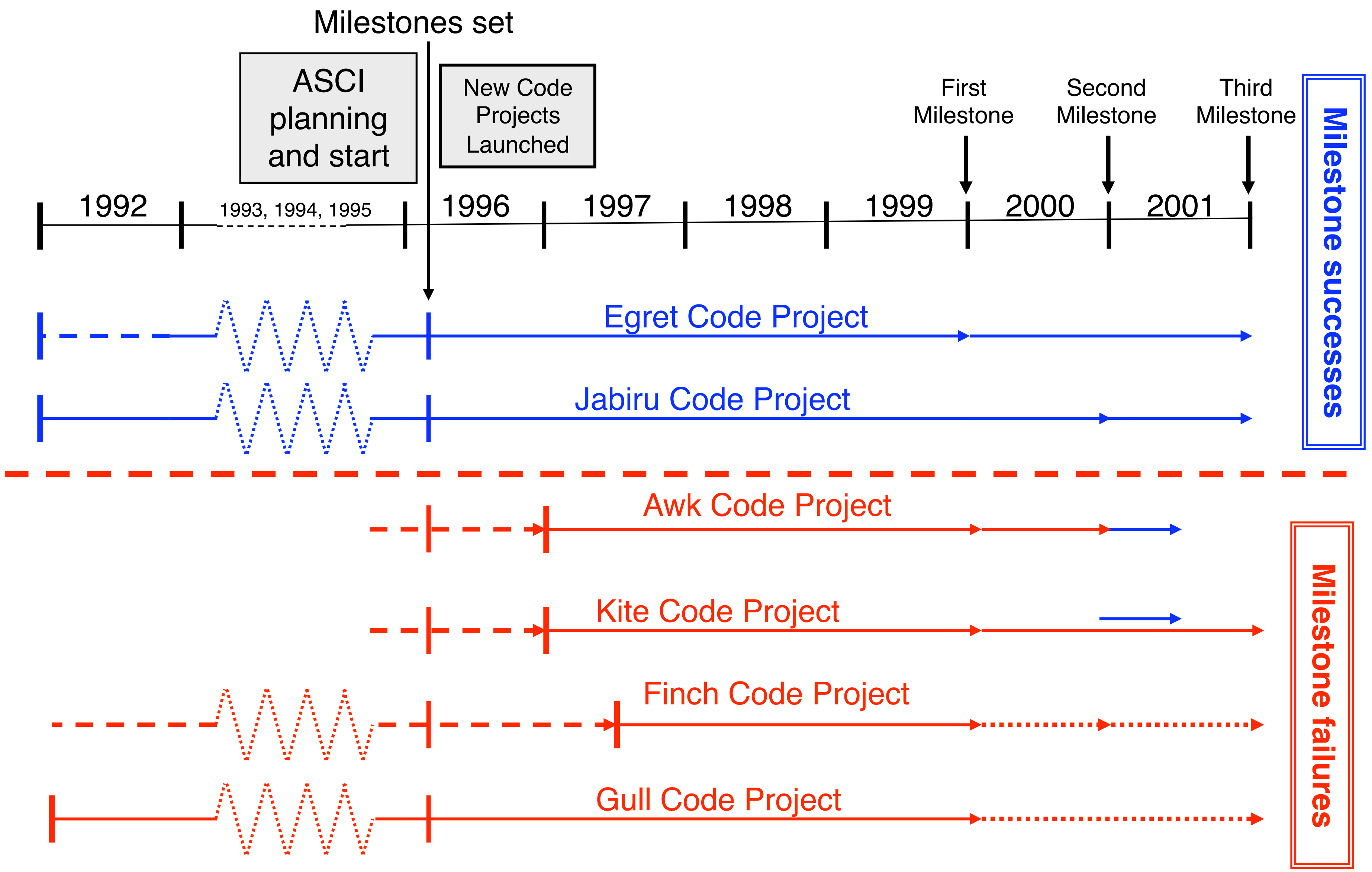
Productivity

- P = Utility/Cost
- Conceptually a great idea – value per unit cost increases => per capita wealth increases
  - Renormalized
  - Inflation
  - Definition of value
- What does it mean for computational science?

ASCI

- In late 1996, the DOE launched the Accelerated Strategic Computing Initiative (ASCI) to develop the“enhanced” predictive capability by 2004 at LANL, LLNL and SNL that was required to certify the US nuclear stockpile without testing
  - ASCI codes were to have much better physics, better resolution and better materials data
  - Need a 10^5 increase in computer power from 1995 level
  - Develop massively parallel platforms (20 TFlops at LANL this year, 100 TFlops at LLNL in 2005-2006)
  - ASCI included development of applications, development and analysis tools, massively parallel platforms, operating and networking systems and physics models
- ~ $6 B expended so far
- First milestones were Jan.1,2000 and Jan.1,2001—3 to 4 years after project start!!!
- Success required development and integration of several major physics capabilities

ASCI Milestone and Code Project History



Egret Code Project was conservative—schedule and technology—and successful

- Originally started in 1992 as part of a graduate thesis building on a prior code
  - Serial version working with all but one of the major required packages in 1994, parallelization began in 1997
- Written in C by a very experienced team of physicists and computer scientists
  - Management experienced with code development
- Early customer involvement (~1995), fairly continuous V&V since 1996
- Began implementation of next major package in 1997
- Strong support by management, well-defined project structure
- Completed first major milestone Dec 28, 1999, three days before Jan. 1, 2000 milestone due date

Jabiru code project was conservative and successful

- Jabiru code project began as a code port from a vendor with all but one of necessary packages (serial) in 1992
  - Installation of other packages begun in 1997
  - Parallelization started in 1994
- Written in Fortran 90
  - Heavy emphasis on portability, (~ 10 different platforms)
  - Conservative, perhaps overly conservative computer science
- Continuous interaction and use by users, long V&V history
- Substantial use by non-ASCI projects (Labs, universities, and other institutions)
- Senior management support polarized, some strongly in favor—”it really works”, some strongly opposed (“not invented here syndrome”)
- Strong example of a “hero” model success, highly experienced and mature staff

Awk Code Project was a very ambitious project

- The Awk Code Project Vision involved using object oriented languages(C++) with a Python controller
- Project began in late 1996
- Very few experienced staff, all very bright but fairly new to the physics and the programming challenge
  - Leadership very young and inexperienced, but smart and dedicated
  - Lots of computer science and programming support
  - Senior management very inexperienced managing code development
- Ambitious physics scope, trying new algorithms that needed considerable R&D
- Initially didn't succeed in meeting milestones—initial milestone not important for customers—customers weren't interested
- Still not a really mature code after 7 years
- Performance a major issue (10 times slower than comparable F90 codes)

Kite Code Project has had mixed successes

- Kite Code Project used Fortran, but with home grown object programming implementation
- Project launched in mid-1996
- Fairly innovative physics approach
- Little programming/computer science support
- Good support from users
- Mixture of experience levels and code development skills
  - Leadership inexperienced with large code groups
  - Senior management inexperienced with successful code development
  - Algorithm support groups failed to deliver essential modules
- Major changes in the milestone requirements were made by DOE six months before milestone was due that invalidated the strategy of the team to complete the milepost
- Didn't complete milestone on time Dec. 31, 1999

Finch Code Project was a shotgun marriage that ended in divorce

- Finch Code Project began in early 1997
- Initial project consisted of two major groups in other divisions with existing codes that had lost their support plus an integration group
  - These groups viewed ASCI as a way to get support for their existing work
  - Success was measured by the success of each sub-project individual rather than the success of the whole project
  - The senior management in the other divisions was much more interested in the continued flow of the money than in the success of the Finch project
  - Little or no analysis was done to establish that the two packages were compatible, i.e. that it was technically feasible to integrate the two packages into a single code
  - Finch Project manager had almost no authority over the staff in each sub-project
- The Finch Project never really got off the ground.
  - It didn't meet any of its milestones
  - But it had the best SQA of all of the codes!
- The Finch Project was canceled last year.

Gull Code Project was canceled after ~ $100M

- The Gull Code Project was begun in 1992
- It was based on a F90 parallel code obtained from another laboratory
- The Gull project staff decided to use the project as a vehicle for improving code development methodologies
  - Automatic and general parallelization through a library
  - Latest object oriented computer languages (C++)
  - Automatic differencing and code generation from “equations”
  - Performance issues: Substantially outperformed by legacy code that it was based on (which only had about 1 FTE of support per year)
- Very large code team (~ 50 staff) spread across 3 divisions without much authority given to project lead
  - Project leaders were often inexperienced and there was substantial project leadership and staff turnover
- Hostility toward customers (“We know what you need better than you do!”)
- Failed to meet ultimate milestones (although a few intermediate ones were met)
- Canceled last year after $100M (about expending about one-half of the weapons code development resources at one of the labs)

We have used these results to identify some *“Lessons Learned”*

The Successful projects emphasized:

- Building on successful code development history and prototypes
- Highly competent and motivated people in a good team
- Risk identification, management and mitigation
- Software Project Management: Run the code project like a project
- Determine the Schedule and resources from the requirements
- Customer focus
  - For code teams and for stakeholder support
- Better physics and computational mathematics is much more important than better “computer science”
- The use of modern but proven Computer Science techniques,
  - They don’t make the code project a Computer Science research project
- Develop the team
- Software Quality Engineering: Best Practices rather than Processes
- Validation and Verification

The unsuccessful projects didn’t emphasize these!

The Lessons Learned are simple.

- Build on successful code development history and prototypes
  - Start with successes and evolve from them
  - LANL and LLNL had a record of small team successes
    - Moved quickly to large teams wth the result that there was a lot of stumbling and turmoil
- Highly competent and motivated people in a good team
  - It's all about building and supporting a good team.
  - Finch and Gull Code Projects didn't have cohesive teams, the Egret and Jabiru projects did
- Risk identification, management and mitigation
  - LANL and some of LLNL management blamed the code teams for failures due to poor planning, inadequate support and other things beyond the control of the team
  - Most failures are due to lack of management support and constructive oversight
  - Finch code project risks were not analyzed and dealt with
- Software Project Management: Run the code project like a project
  - If the code project leaders are to be accountable, they need authority
  - The Gull and Finch project leaders didn't have the authority they needed to run their projects, the Egret and Jabiru project leaders did
  - Give authority to those responsible for the project
  - A research program is not a project! Proper balance between R&D and structured development is essential